GNNs for Recommendation Systems

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

Contents

- Tasks and Evaluation metrices
- Embedding models
- GNNs for Recommendation systems
  - Traditional Collaborative Filtering
  - Neural Graph Collab. Filtering (NGCF)
  - LightGCN
  - Recommender Systems With Knowledge Graph

Preliminary of Recommendation

- **Information Explosion in the era of Internet**
  - 10K+ movies in Netflix
  - 12M products in Amazon (350m on Marketplace)
  - 70M+ music tracks in Spotify
  - 10B+ videos on YouTube
  - 200B+ pins (images) in Pinterest
- **Personalized recommendation**
  - i.e., suggesting a small number of interesting items for each user) is critical for users to effectively explore the content of their interest.

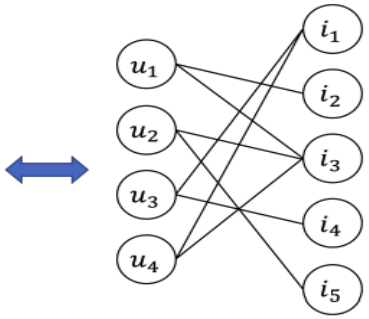
Recommender systems (RS)

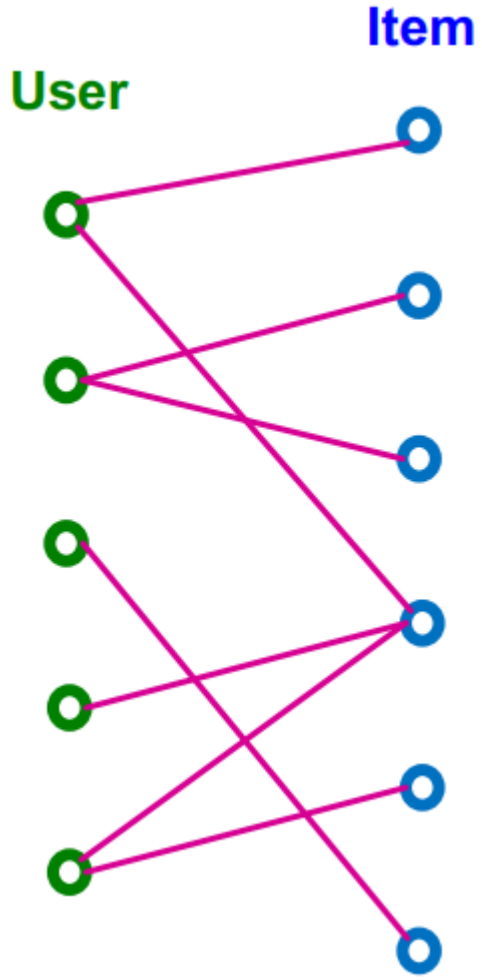
- **What are the inputs of recommender system?**
  - Main info:
    - user (attributes)
    - item (attributes)
    - user feedback on items (user-item interactions)
  - Side info: (to solve data sparsity and cold start issue)
    - social relationship between users
    - knowledge graph
- **Tasks:**
  - users' preferences
  - item properties

Recommender System as a Graph

- Recommender system can be naturally modeled as a **bipartite graph**
  - A graph with two node types: users and items.
  - **Edges** connect **users** and **items**
    - Indicates user-item interaction (e.g., click, purchase, review etc.)
    - Often associated with timestamp (timing of the interaction).

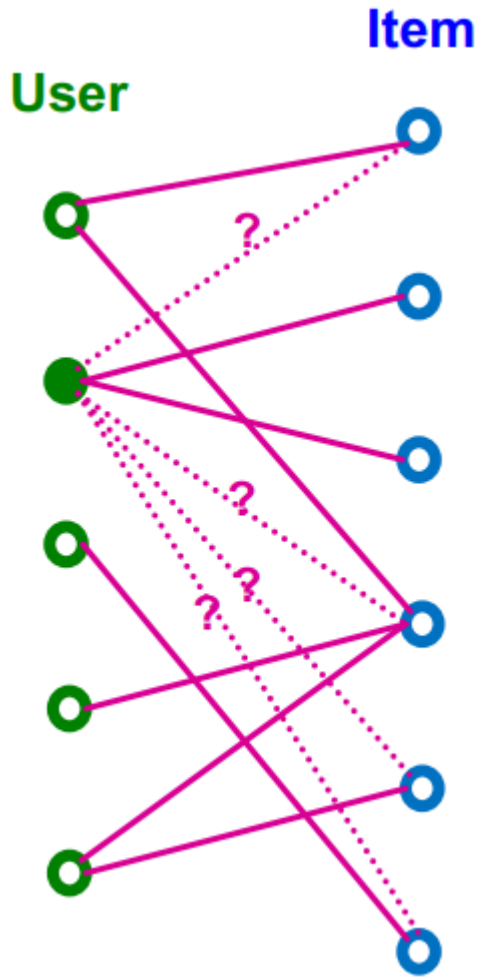
	i_1	i_2	i_3	i_4	i_5
u_1	0	1	1	0	0
u_2	0	0	1	0	1
u_3	1	0	0	1	0
u_4	1	0	1	1	1





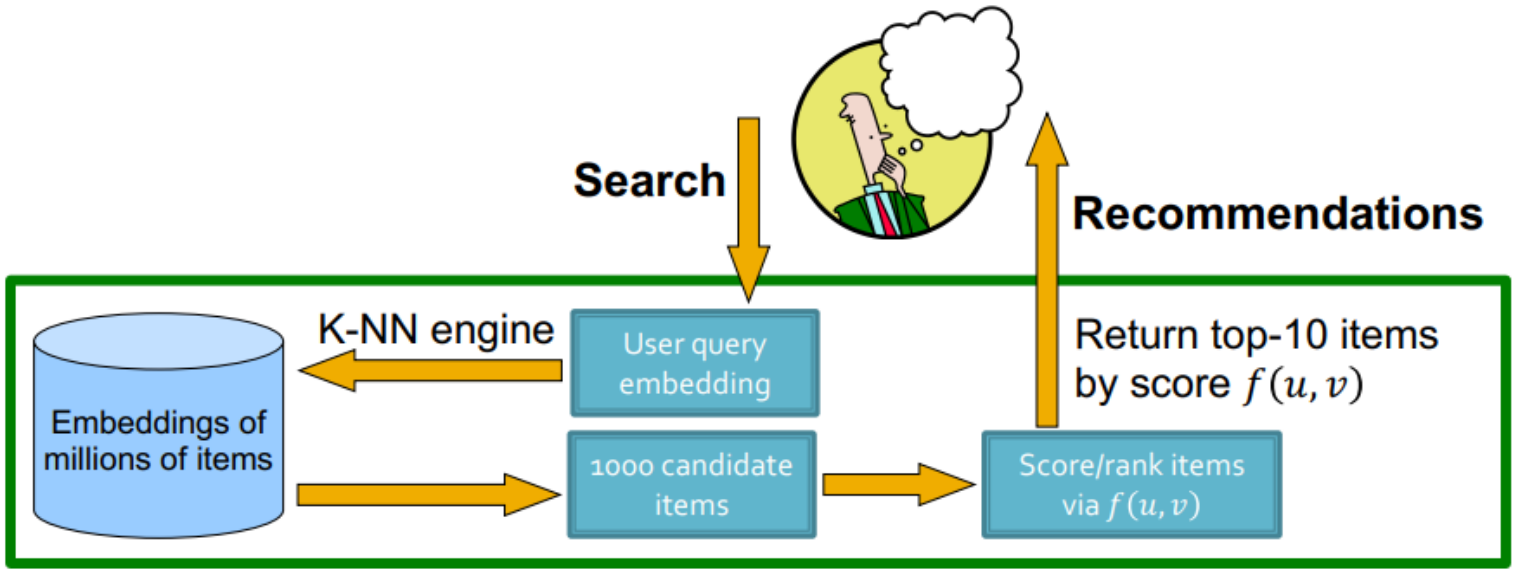
Recommender System as a Graph

- **Given**:
  - Past user-item interactions
- **Task**:
  - Predict new items each user will interact in the future.
  - Can be cast as link prediction problem.
    - Predict new user-item interaction edges given the past edges.
  - For $u \in U, v \in V$, we need to get a real-valued score $f(u, v)$



- Problem: Cannot evaluate $f(u, v)$ for every user u – item v pair.
- Solution: 2-stage process:
  - Candidate generation (cheap, fast)
  - Ranking (slow, accurate)

Example $f(u, v)$:
$f(u, v) = z_u \cdot z_v$

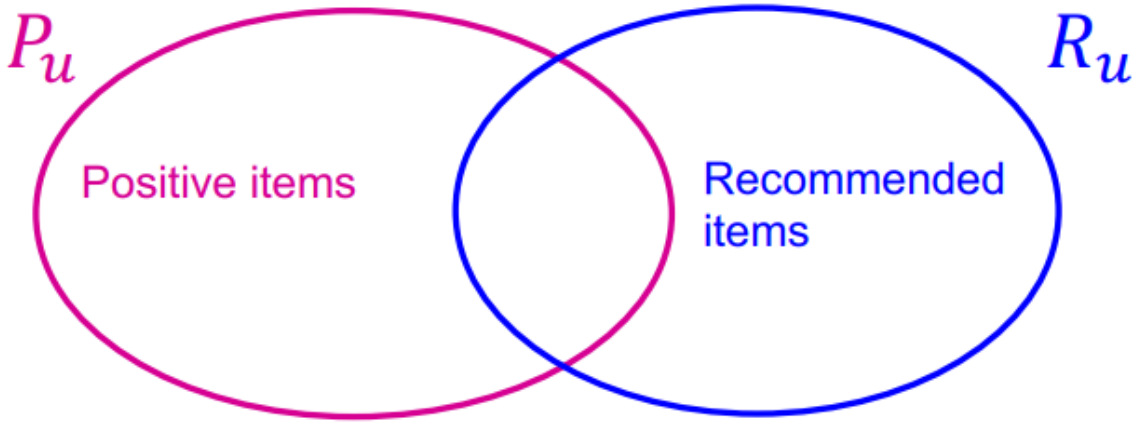


Top-*K* Recommendation

- For each user, we recommend K items.
  - For recommendation to be effective, K needs to be much smaller than the total number of items (up to billions)
  - K is typically in the order of 10—100.
- The goal is to include as many positive items as possible in the top-*K* recommended items.
  - Positive items = Items that the user will interact with in the future.
- Evaluation metric: Recall@*K*

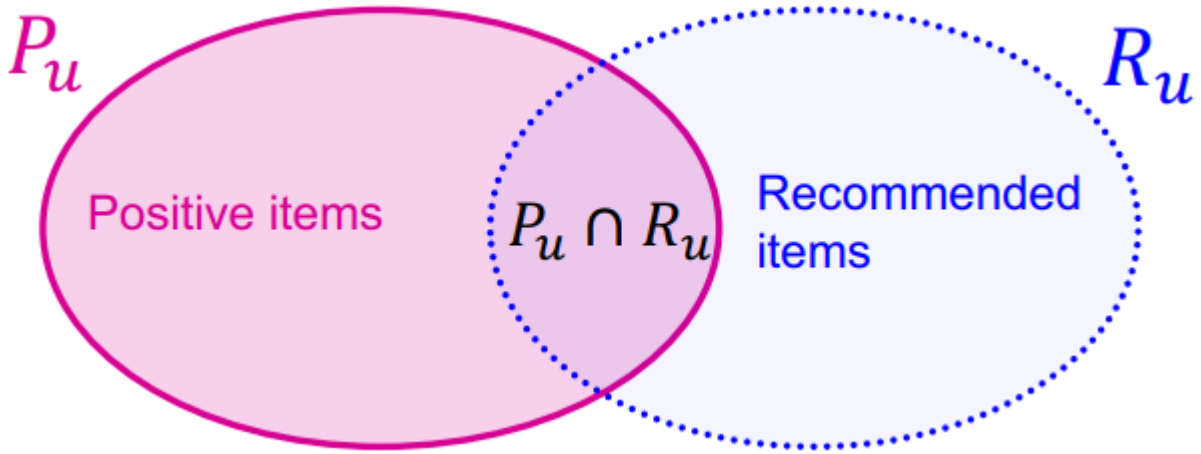
Evaluation Metric: Recall@k

- For each user u:
  - Let P_u be a set of positive items the user will interact in the future.
- Let R_u be a set of items recommended by the model.
  - In top-K recommendation, $|R_u|$ = K.
  - Items that the user has already interacted are excluded.



Evaluation Metric: Recall@k

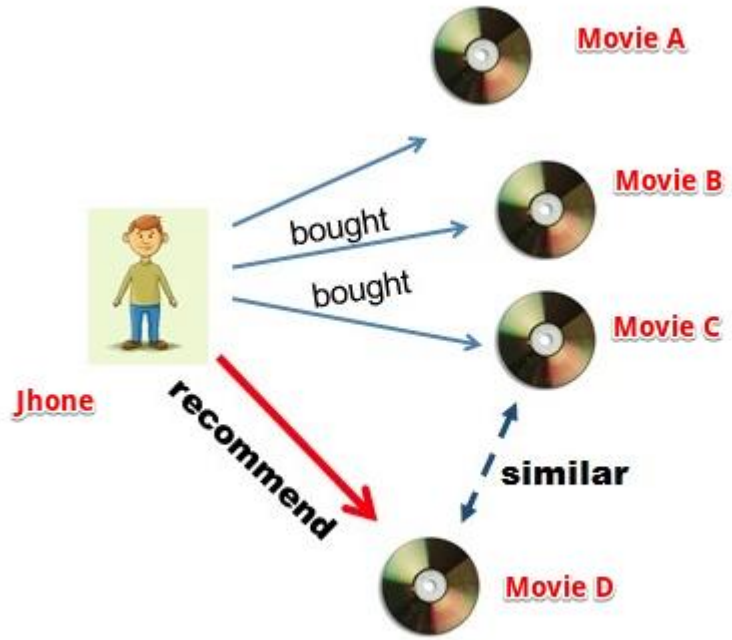
- Recall@K for user u is $|\boldsymbol{P_u} \cap \boldsymbol{R_u}|/|\boldsymbol{P_u}|$
- Higher value indicates more positive items are recommended in top-K for user u.



- The final Recall@K is computed by averaging the recall values across all users.

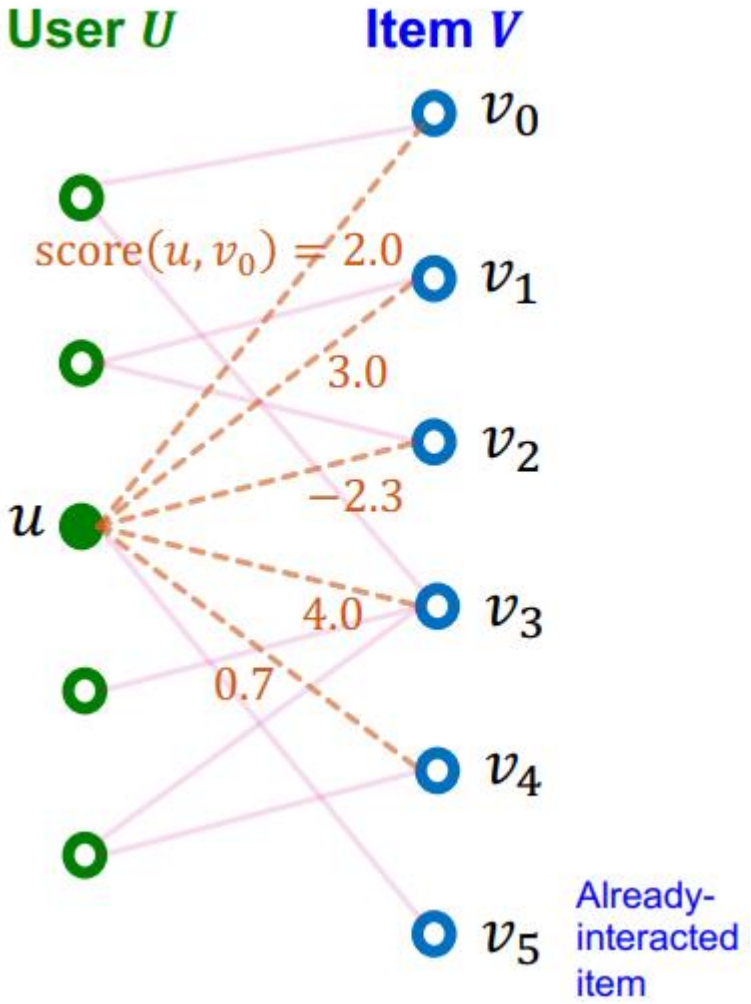
- $\boldsymbol{U}$: A set of all users
- $\boldsymbol{V}$: A set of all items
- $\boldsymbol{E}$: A set of observed user-item interactions

$$\boldsymbol{E} = \{(u, v) \mid u \in \boldsymbol{U}, v \in \boldsymbol{V}, u \text{ interacted with } v\}$$



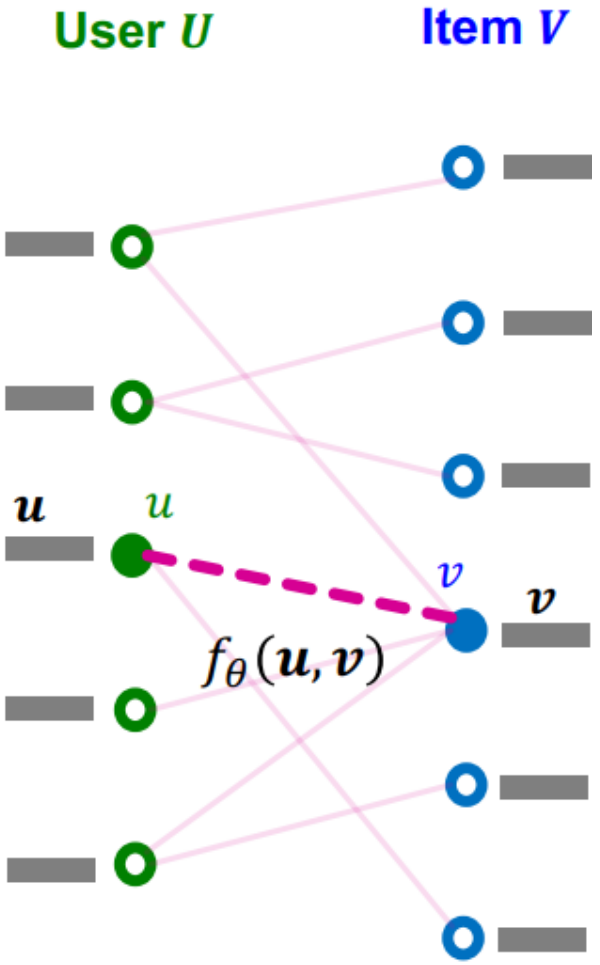
Score Function

- To get the top-K items, we need a score function for user-item interaction:
  - For $u \in U, v \in \boldsymbol{V}$, we need to get a real-valued scalar score(u, v).
  - K items with the largest scores for a given user u (excluding alreadyinteracted items) are then recommended
  - i.e., For K = 2, recommended items for user u would be v_1, v_3.



Score Function

- We consider **embedding-based models** for scoring user-item interactions.
  - For each user $u \in U$, let $u \in R^d$ be its *d*-dimensional embedding.
  - For each item $v \in V$, let $v \in R^d$ be its *d*-dimensional embedding.
  - Let $f_\theta(\cdot,\cdot)$: $R^d \times R^d \rightarrow R$ be a parametrized function.
  - Then, $\text{score}(u, v) \equiv f_\theta(\boldsymbol{u}, \boldsymbol{v})$



Training Objective

- Embedding-based models have three kinds of parameters:
  - An encoder to generate user embeddings {u}
  - An encoder to generate item embeddings {v}
  - Score function $f_\theta(\cdot,\cdot)$

- Training objective: Optimize the model parameters to achieve high recall@*K* on seen (i.e., training) user-item interactions
  - We hope this objective would lead to high recall@*K* on unseen (i.e., test) interactions

Loss Functions

- The original training objective (recall@*K*) is not differentiable.
  - Cannot apply efficient gradient-based optimization.

- Two surrogate loss functions are widely-used to enable efficient gradient-based optimization.
  - Binary loss
  - Bayesian Personalized Ranking (BPR) loss

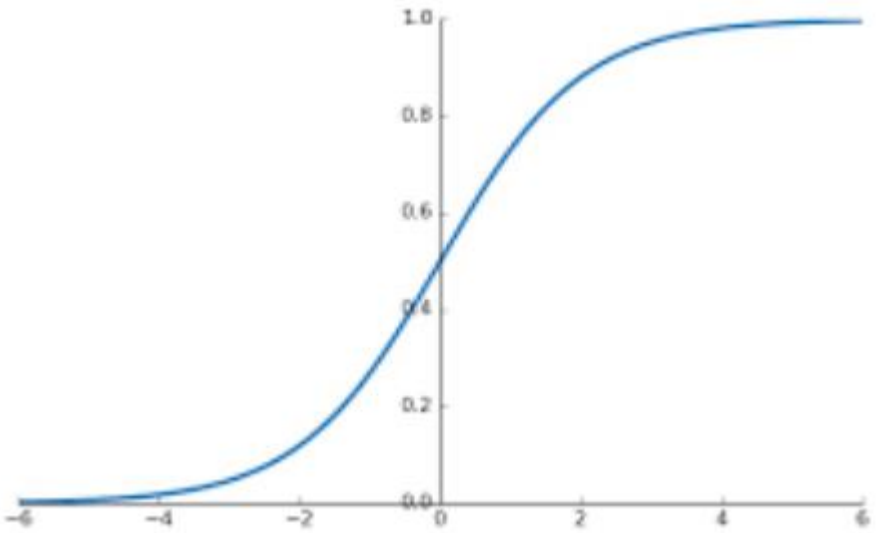
- Surrogate losses are differentiable and should align well with the original training objective.

Binary Loss

- Define positive/negative edges
- A set of **positive edges E** (i.e., observed/training user-item interactions)
- A set of **negative edges** $E_{neg} = \{(u,v)|(u,v) \notin E, u \in U, v \in V\}$
- Define sigmoid function

$$\sigma(x) \equiv \frac{1}{1+\exp(-x)}$$

  - Maps real-valued scores into binary likelihood scores, i.e., in the range of [0,1].



Binary Loss

- Binary loss: Binary classification of positive/negative edges using $\sigma(f_\theta(\boldsymbol{u}, \boldsymbol{v}))$:

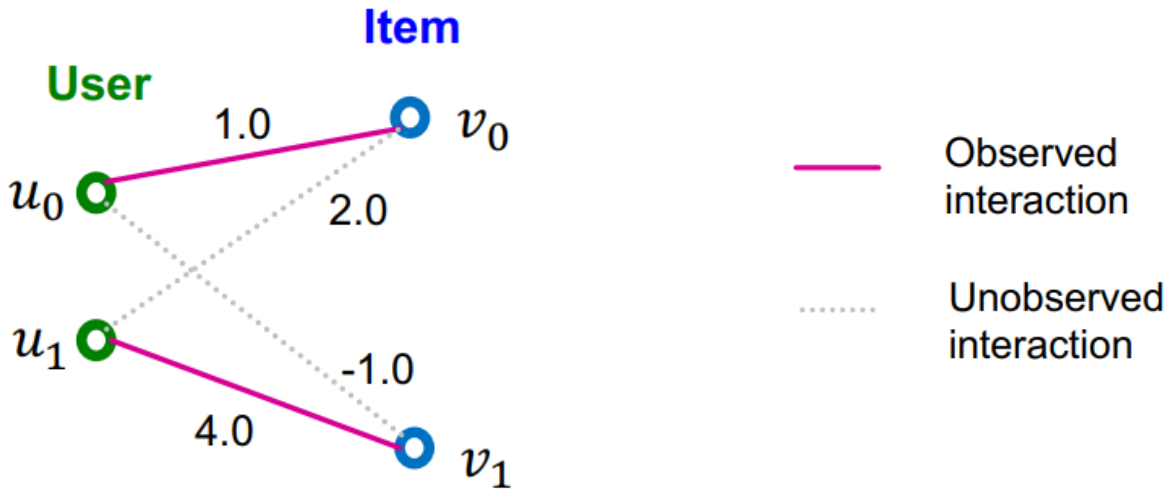
$$\underbrace{-\frac{1}{|\boldsymbol{E}|}\sum_{(u,v)\in \boldsymbol{E}} \log\left(\sigma(f_\theta(\boldsymbol{u}, \boldsymbol{v}))\right)}_{} \underbrace{-\frac{1}{|\boldsymbol{E}_{\text{neg}}|}\sum_{(u,v)\in \boldsymbol{E}_{\text{neg}}} \log\left(1-\sigma(f_\theta(\boldsymbol{u}, \boldsymbol{v}))\right)}_{}$$

During training, these terms can be approximated using mini-batch of positive/negative edges

- Binary loss pushes the scores of positive edges higher than those of negative edges.
  - This aligns with the training recall metric since positive edges need to be recalled.

Desirable Surrogate Loss

- Surrogate loss function should be defined in a personalized manner.
  - For each user, we want the scores of positive items to be higher than those of the negative items
  - We do not care about the score ordering across users.
- Bayesian Personalized Ranking (BPR) loss achieves this!

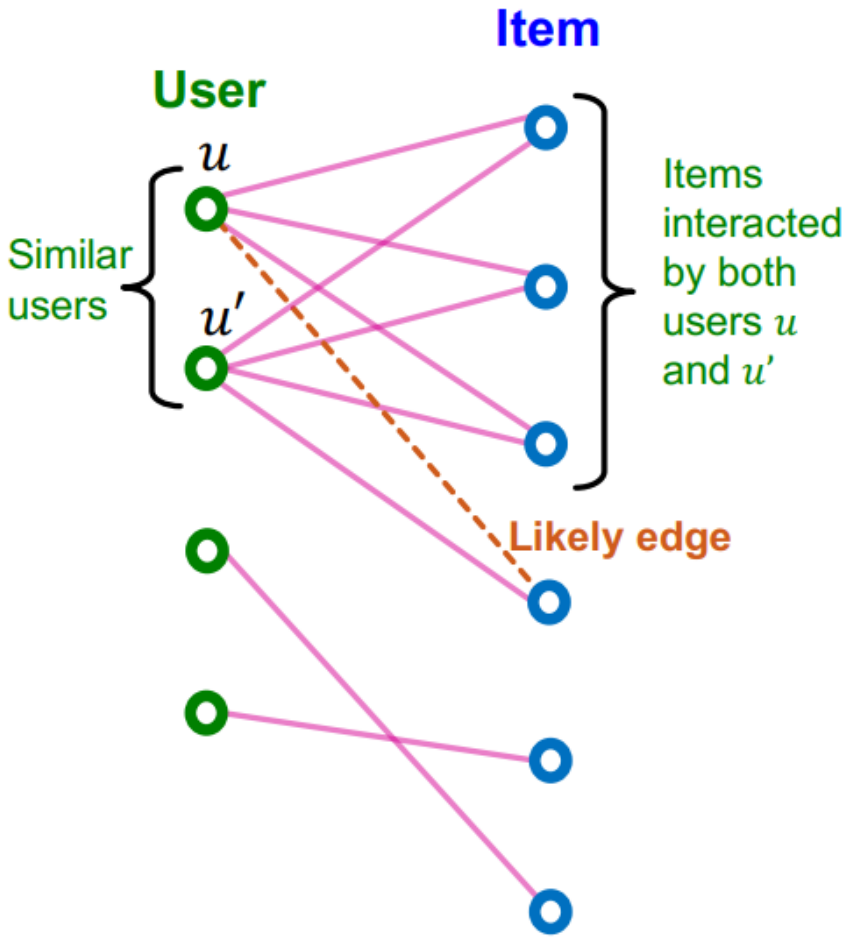


Summary so far

- We have introduced
  - Recall@*K* as a metric for personalized recommendation
  - Embedding-based models
    - Three kinds of parameters to learn
      - user encoder to generate user embeddings
      - item encoder to generate item embeddings
      - score function to predict the user-item interaction likelihood.
  - Surrogate loss functions to achieve the high recall metric.
- Embedding-based models have achieved SoTA in recommender systems.
  - Why do they work so well?

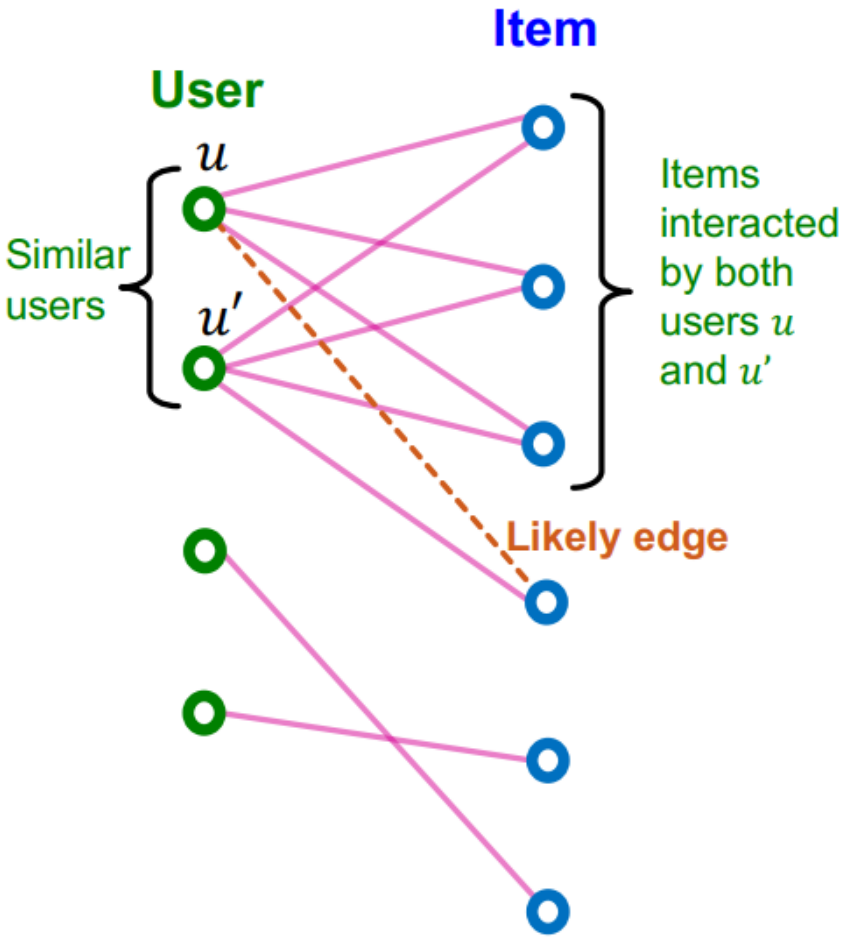
Why Embedding Models work?

- **Underlying idea**: Collaborative filtering
  - Recommend items for a user by collecting preferences of many other similar users.
  - Similar users tend to prefer similar items.
- **Key question**: How to capture similarity between users/items?



Why Embedding Models work?

- Embedding-based models can capture similarity of users/items
  - Low-dimensional embeddings cannot simply memorize all user-item interaction data.
  - Embeddings are forced to capture similarity between users/items to fit the data.
  - This allows the models to make effective prediction on unseen user-item interactions.



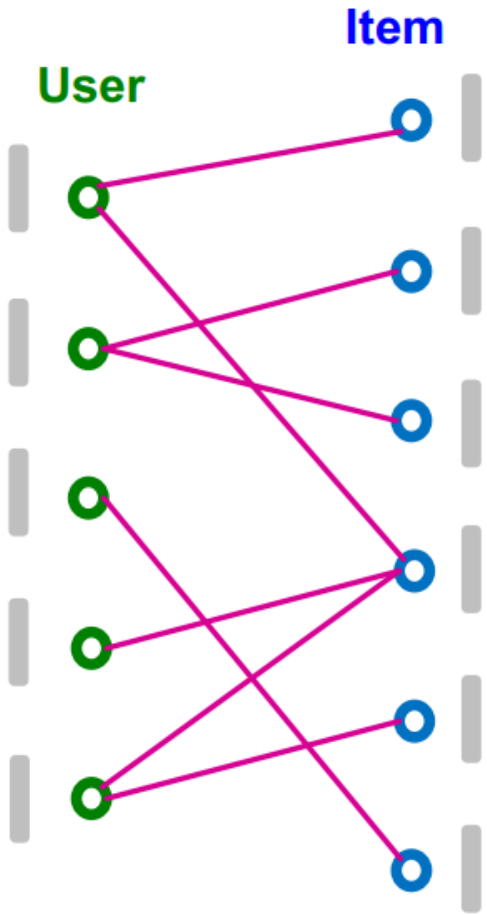
- Two representative GNN approaches for recommender systems:
  - (1) Neural Graph Collab. Filtering (NGCF) [Wang et al. 2019]
  - (2) LightGCN [He et al. 2020]
    - Improve the conventional collaborative filtering models (i.e., shallow encoders) by explicitly modeling graph structure using GNNs.
    - Assumes no user/item features.
  - (3) KGAT [Wang et al. 2019]

Conventional Collaborative Filtering

- Conventional collaborative filtering model is based on shallow encoders:
  - No user/item features.
  - Use shallow encoders for users and items

- Score function for user u and item i is:

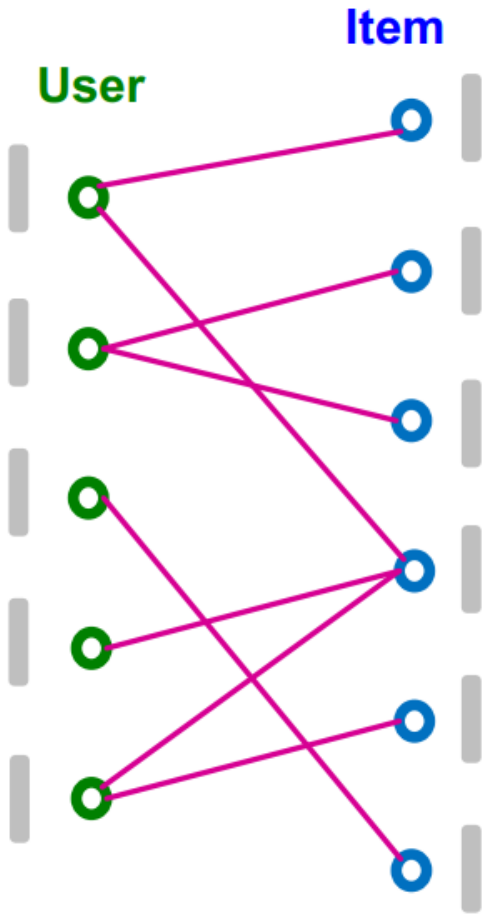
$$f_\theta(u, v) \equiv z_u{}^T z_v.$$



Learnable shallow user/item embeddings

Conventional Collaborative Filtering: Limitations

- The model itself does not explicitly capture graph structure
  - The graph structure is only implicitly captured in the training objective.
  - Only the first-order graph structure (i.e., edges) is captured in the training objective.
  - High-order graph structure (e.g., *K*-hop paths between two nodes) is not explicitly captured.



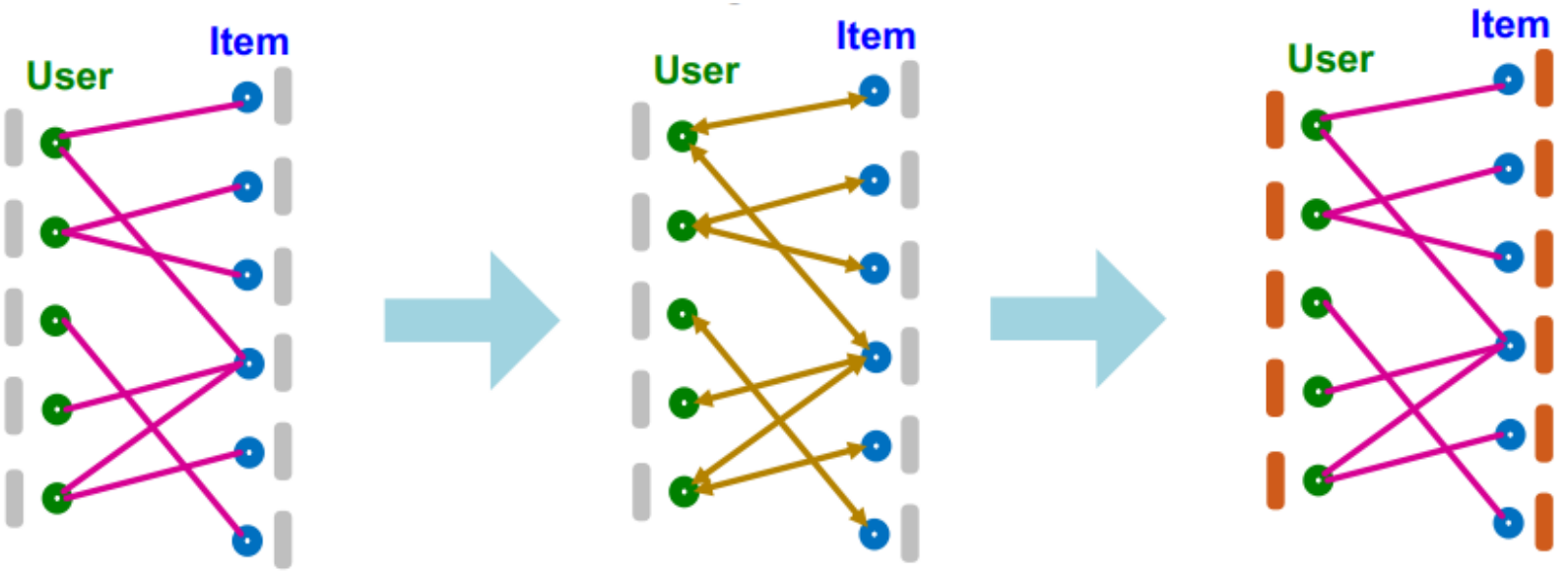
Learnable shallow user/item embeddings

Motivation

- We want a model that:
  - explicitly captures graph structure (beyond implicitly through the training objective)
  - captures high-order graph structure (beyond the first-order edge connectivity structure)
- GNNs are a natural approach to achieve both
  - Neural Graph Collaborative Filtering (NGCF) [Wang et al. 2019]
  - LightGCN [He et al. 2020]
    - A simplified and improved version of NGCF

NGCF overview

- Neural Graph Collaborative Filtering (NGCF) explicitly incorporates high-order graph structure when generating user/item embeddings.
- **Key idea**: Use a GNN to generate graph-aware user/item embeddings.



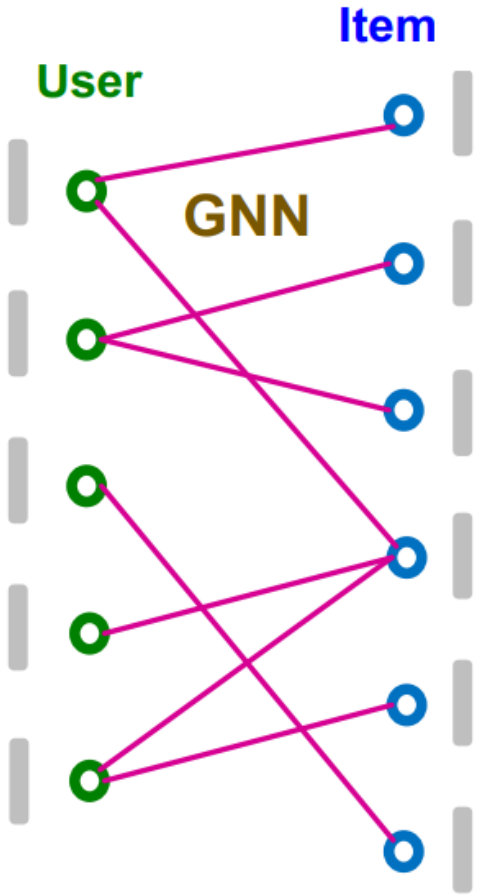
Initial shallow embeddings (not graph-aware)

Use a GNN to propagate embeddings

NGCF's graph-aware embeddings

NGCF Framework

- Given: User-item bipartite graph.
- NGCF framework:
  - Prepare shallow learnable embedding for each node.
  - Use multi-layer GNNs to propagate embeddings along the bipartite graph.
    - High-order graph structure is captured.
  - Final embeddings are explicitly graphaware
- Two kinds of learnable params are jointly learned:
  - Shallow user/item embeddings
  - GNN's parameters



Shallow user/item embeddings (learnable)

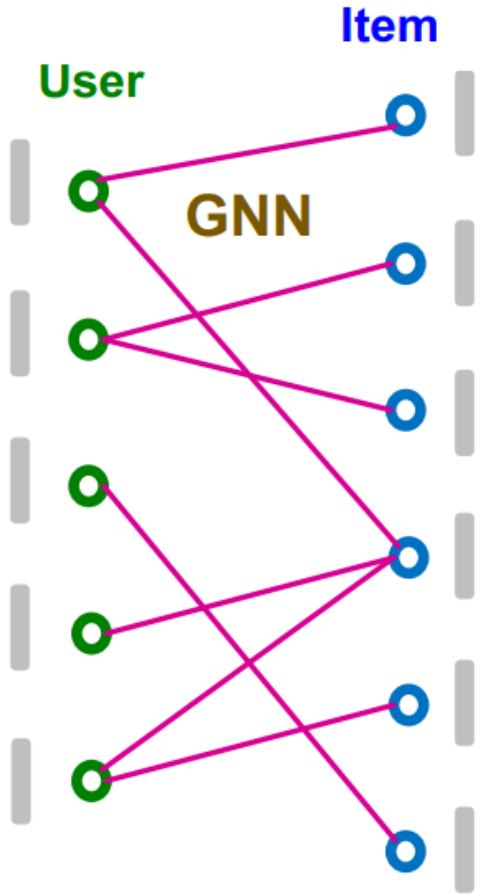
NGCF Framework: Initial node embeddings

- Set the shallow learnable embeddings as the initial node features:
  - For every user $u \in U$, set h_u^0 as the user's shallow embedding.
  - For every item $v \in V$, set h_v^0 as the item's shallow embedding.
- **Neighbour aggregation:**
  - Iteratively update node embeddings using neighboring embeddings.

$$\boldsymbol{h}_v^{(k+1)} = \text{COMBINE}\left(\boldsymbol{h}_v^{(k)}, \text{AGGR}\left(\left\{\boldsymbol{h}_u^{(k)}\right\}_{u \in N(v)}\right)\right)$$

$$\boldsymbol{h}_u^{(k+1)} = \text{COMBINE}\left(\boldsymbol{h}_u^{(k)}, \text{AGGR}\left(\left\{\boldsymbol{h}_v^{(k)}\right\}_{v \in N(u)}\right)\right)$$

  - High-order graph structure is captured through iterative neighbor aggregation.



Shallow user/item embeddings (learnable)

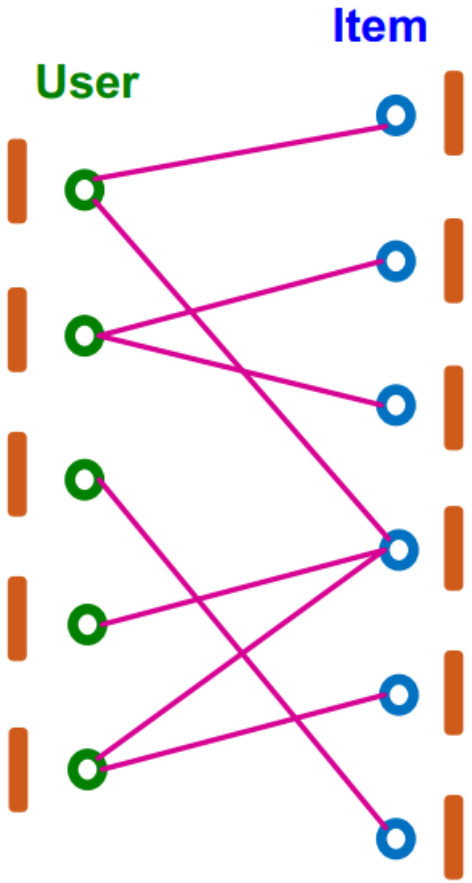
NGCF Framework: Final Embeddings

- After k rounds of neighbor aggregation, we get the final user/item embeddings $h_u^{(k)}$ and $h_v^{(k)}$.
- we set

$$\boldsymbol{u} \leftarrow \boldsymbol{h}_u^{(K)}, \boldsymbol{v} \leftarrow \boldsymbol{h}_v^{(K)}.$$

- Score function is the inner product

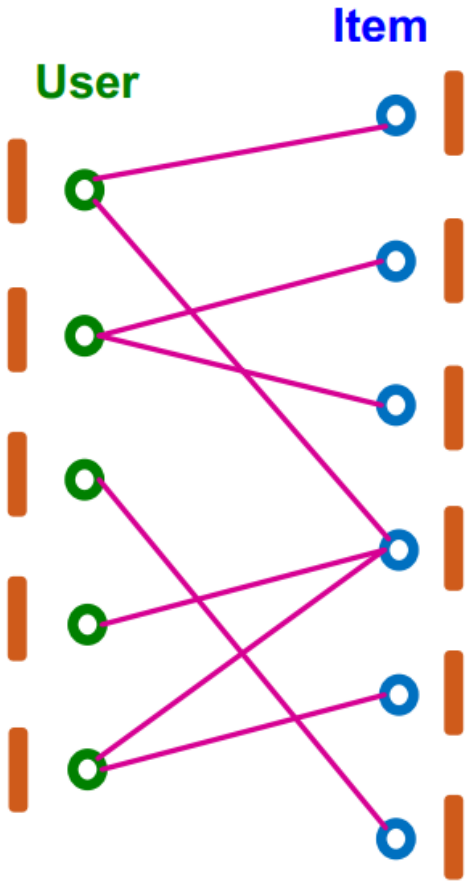
$$\text{score}(u, v) = \boldsymbol{u}^T \boldsymbol{v}$$



Final user/item embeddings (graph-aware)

NGCF: Summary

- Conventional collaborative filtering uses shallow user/item embeddings.
  - The embeddings do not explicitly model graph structure.
  - The training objective does not model high-order graph structure.
- NGCF uses a GNN to propagate the shallow embeddings.
  - The embeddings are explicitly aware of highorder graph structure.



LightGCN: Motivation

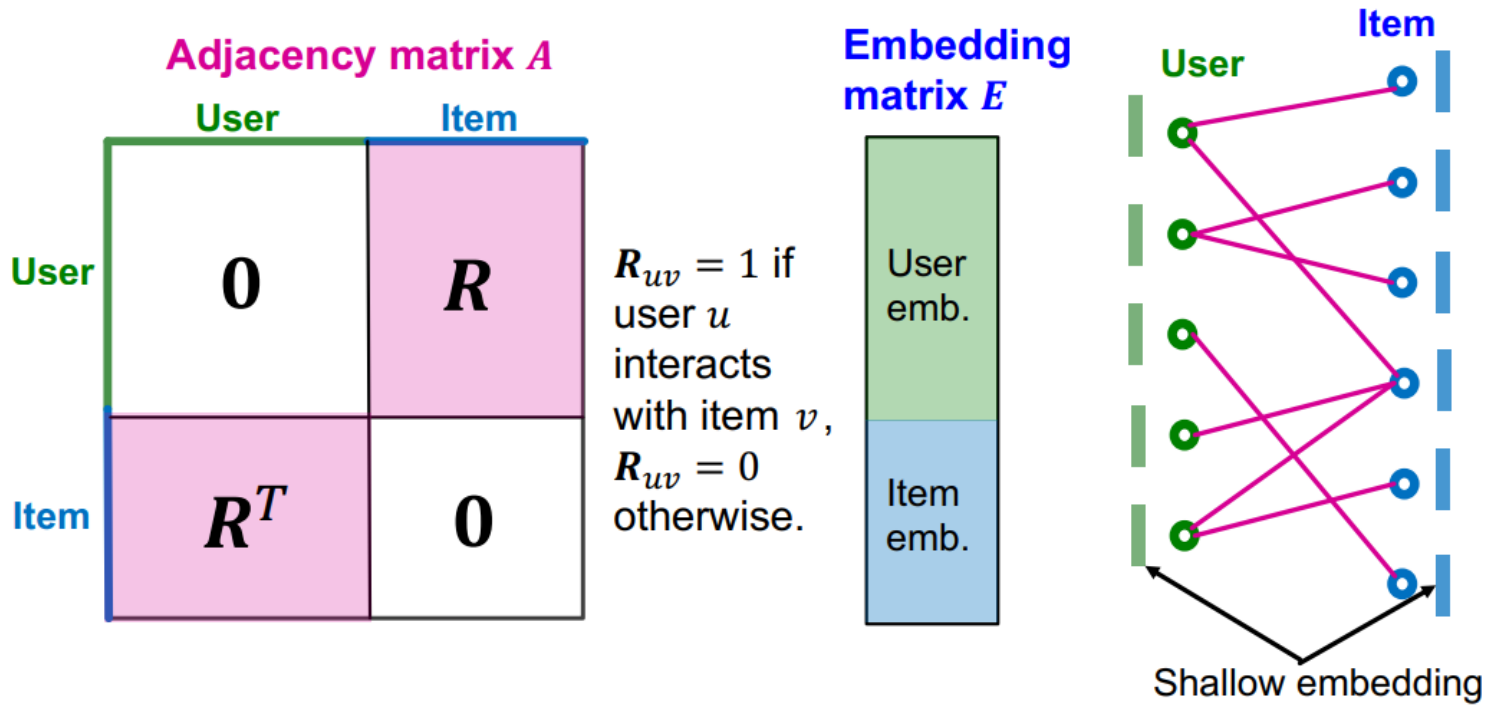
- **Recall**: NGCF jointly learns two kinds of parameters:
  - Shallow user/item embeddings (quite expressive)
  - GNN's parameters

- **Can we simplify the GNN used in NGCF (e.g., remove its learnable parameters)?**

- **Overview of the idea:**
  - Adjacency matrix for a bipartite graph
  - Matrix formulation of GCN
  - Simplification of GCN by removing non-linearity

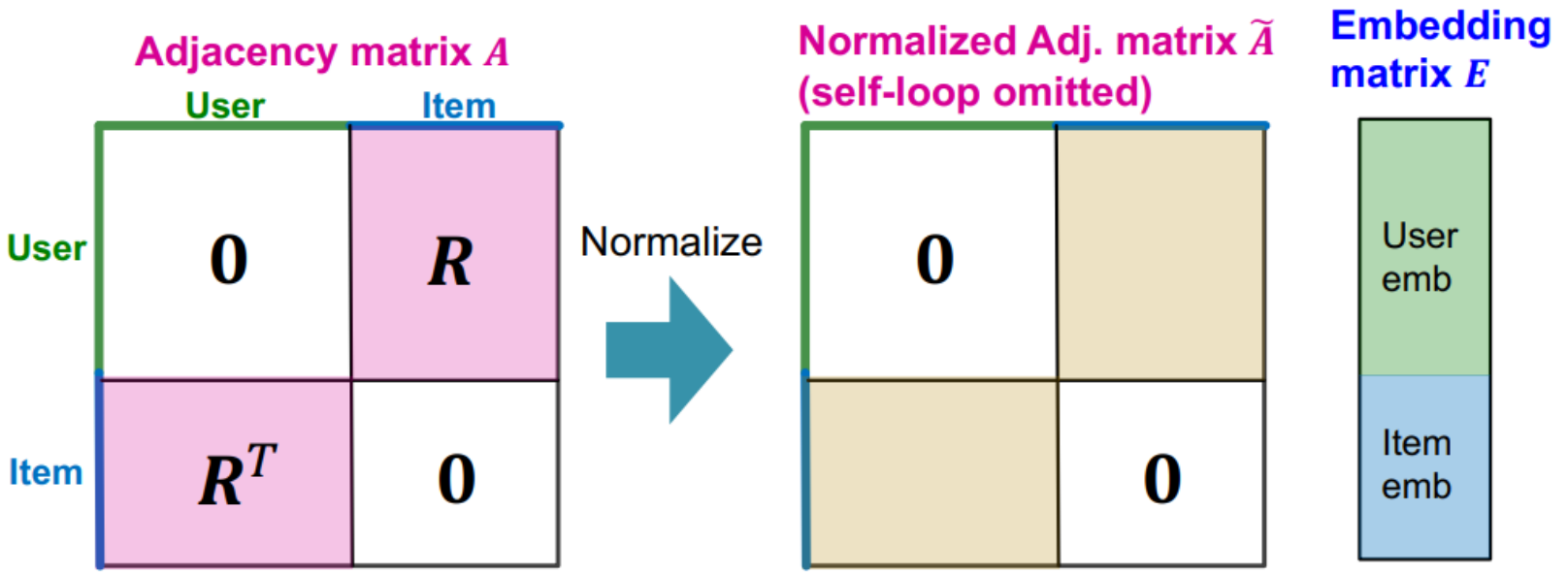
LightGCN: Adjacency & Embedding matrices

- Adjacency matrix of a (undirected) bipartite graph.
- Shallow embedding matrix.



LightGCN: Adjacency & Embedding matrices

- Given:
  - Adjacency matrix A
  - Initial learnable embedding matrix E



LightGCN: Adjacency & Embedding matrices

- **Define**: The diffusion matrix
- Let *D* be the degree matrix of *A*.
- Define the normalized adjacency matrix $\tilde{A}$ as

$$\tilde{\boldsymbol{A}} \equiv \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$$

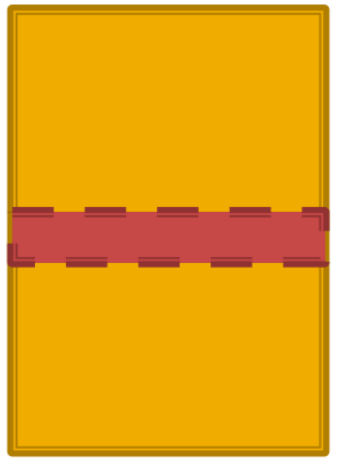
- Let E^k be the embedding matrix at *k*-th layer.
- Each layer of GCN's aggregation can be written in a matrix form:

$$\boldsymbol{E}^{(k+1)} = \mathrm{ReLU}\left(\tilde{\boldsymbol{A}}\boldsymbol{E}^{(k)}\boldsymbol{W}^{(k)}\right)$$

Neighbor aggregation

Learnable linear transformation

Matrix of node embeddings $\boldsymbol{E}^{(k)}$



Each row stores node embedding

LightGCN: Simplifying GCN

- Removing ReLU significantly simplifies GCN

$$\boldsymbol{E}^{(K)} = \widetilde{\boldsymbol{A}}^{K}\,\boldsymbol{E}\,\boldsymbol{W} \qquad \boldsymbol{W} \equiv \boldsymbol{W}^{(0)} \cdots \boldsymbol{W}^{(K-1)}$$

- Diffusing node embeddings along the graph
- Algorithm: Apply $\boldsymbol{E} \leftarrow \widetilde{\boldsymbol{A}}\,\boldsymbol{E}$ for *K* times
- Each matrix multiplication diffuses the current embeddings to their one-hop neighbors.
- Note: $\tilde{A}^{K}$ is dense and never gets materialized. Instead, the above iterative matrix-vector product is used to compute $\widetilde{\boldsymbol{A}}^{K}\,\boldsymbol{E}$.

LightGCN and GCN

- The embedding propagation of LightGCN is closely related to GCN
- Recall: GCN (neighbor aggregation part)

$$\boldsymbol{h}_v^{(k+1)} = \sum_{u \in N(v)} \frac{1}{\sqrt{d_u}\sqrt{d_v}} \cdot \boldsymbol{h}_u^{(k)}$$

Node degree

- Self-loop is added in the neighborhood definition.
- LightGCN uses the same equation except that
  - Self-loop is not added in the neighborhood definition.
  - Final embedding takes the average of embeddings from all the layers:

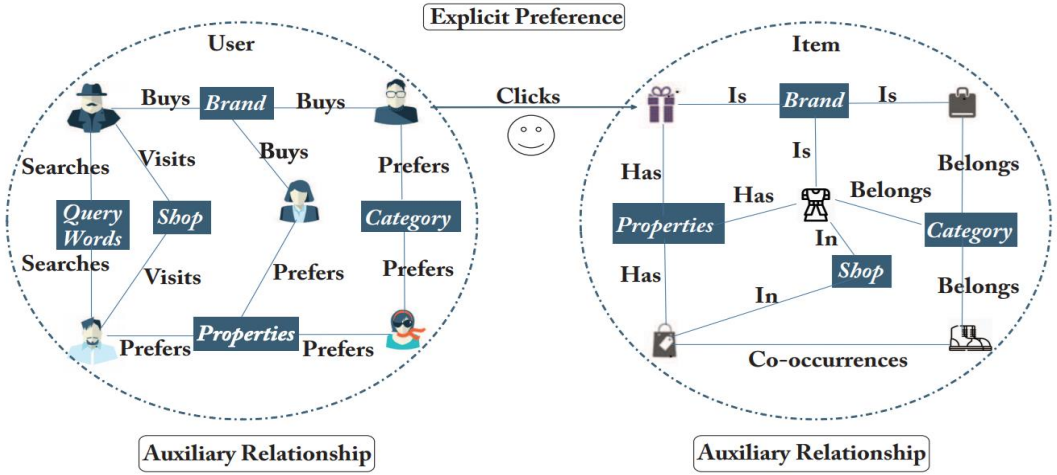
$$\boldsymbol{h}_v = \frac{1}{K+1} \sum_{k=0}^{K} \boldsymbol{h}_v^{(k)}.$$

LightGCN: Summary

- LightGCN simplifies NGCF by removing the learnable parameters of GNNs.
- Learnable parameters are all in the shallow input node embeddings.
  - Diffusion propagation only involves matrix-vector multiplication.
  - The simplification leads to better empirical performance than NGCF.

Recommender Systems With Knowledge Graph

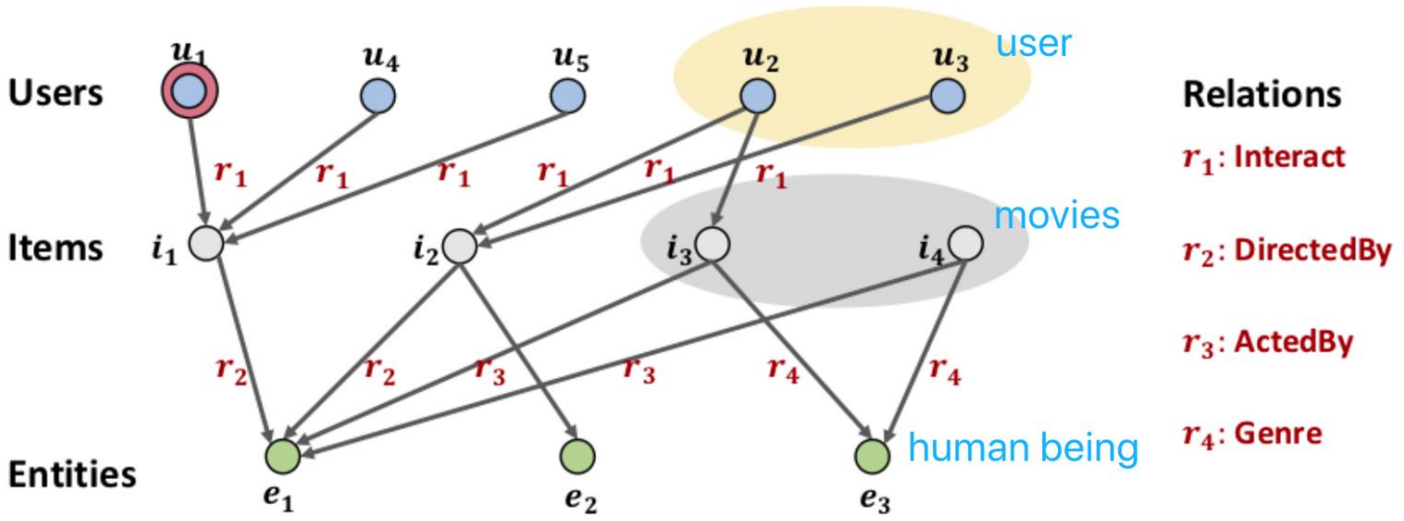
- Knowledge graph:
  - Rich semantic relatedness among items in KG
  - Improve the accuracy of prediction result
  - Can reasonable extend a user's interest



- Difficulty: How to use KG to enhance the prediction?
  - Integration: Treat two graphs separately or merge them?
  - Graph Simplification: Do we make all efforts on the KG, or on its sub-graph?
  - Multi-relation Propagation: There are different relationship. Should we give them the same weight for a certain user?

RecSys With KG: Representative Method

- KGAT:
  - Combine two graphs into one
  - Use attention mechanism by (entity1, relation, entity2)

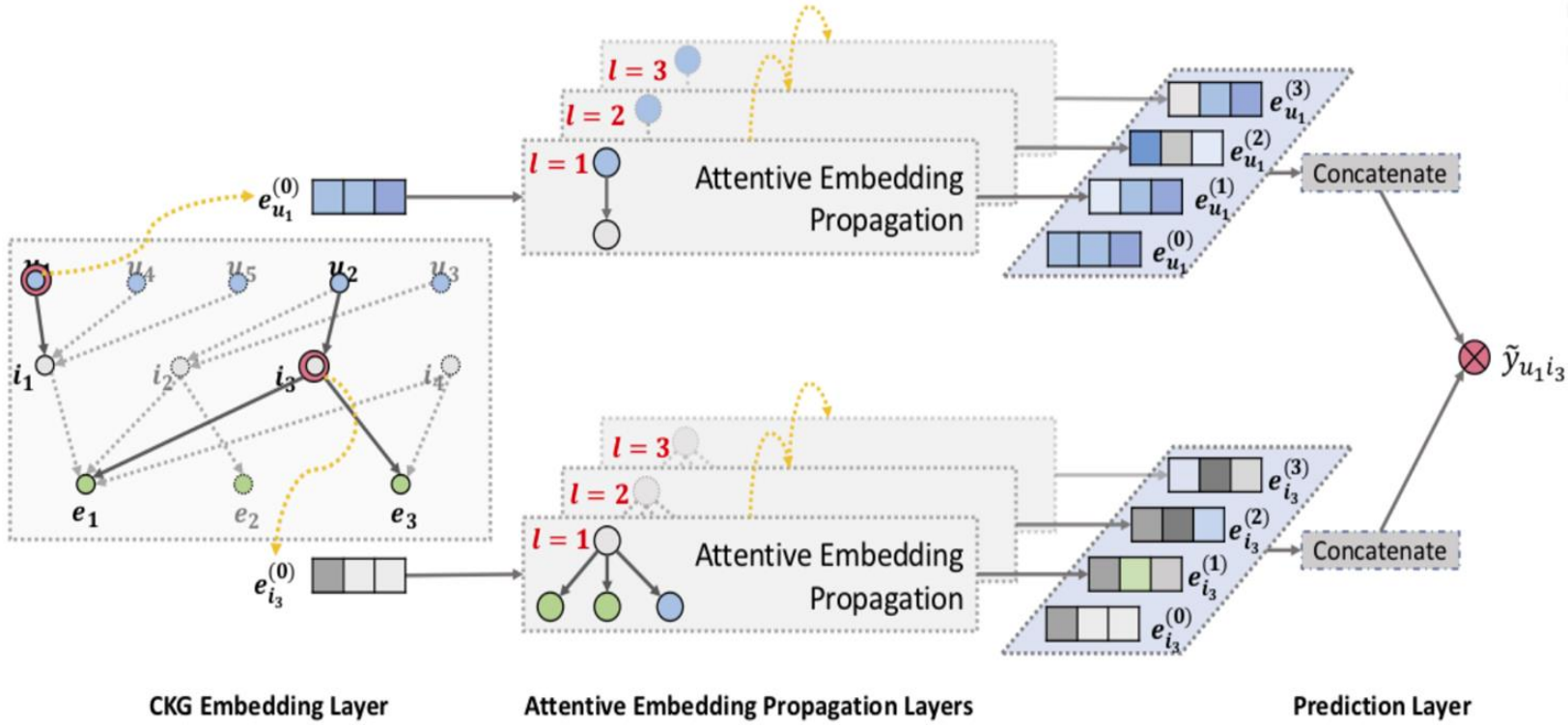


Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD* (pp. 950-958).

- KGAT:
  - For each item and user node, use attention to gather their neibor's information
  - Concatenate embeddings from different jump (l=1, l=2, l=3… l=L)

$$\pi_r^u = g(r, u)$$

$$v_{N(v)}^u = \sum_{e \in N(v)} \pi_{rv,e}^u$$



네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA